
Los Alamos
NATIONAL LABORATORY
EST. 1943


NNSA
National Nuclear Security Administration

Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

NNSA
National Nuclear Security Administration

# FleCSI

a framework designed to support multi-physics application development.


Irina Demeshko

09.01.2020

Los Alamos
NATIONAL LABORATORY
EST.1943



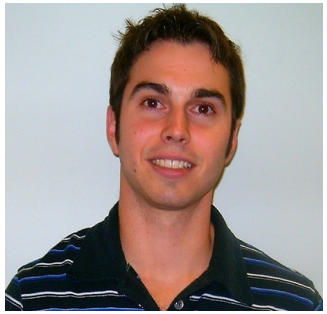
LA-UR-20-26483

# FleCSI team

Ben Bergen

David Daniel

Marc Charest



Andrew Reisner

Karen Tsai

Irina Demeshko

Navamita Ray

Julien Loiseau

Charles Ferenbaugh

Davis Herring

Li-Ta (Ollie) Lo

Jonathan Graham

Christoph Junghans

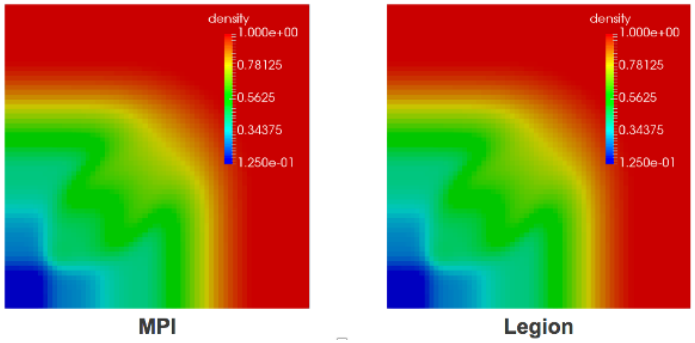
Scot Halverson

# FleCSI

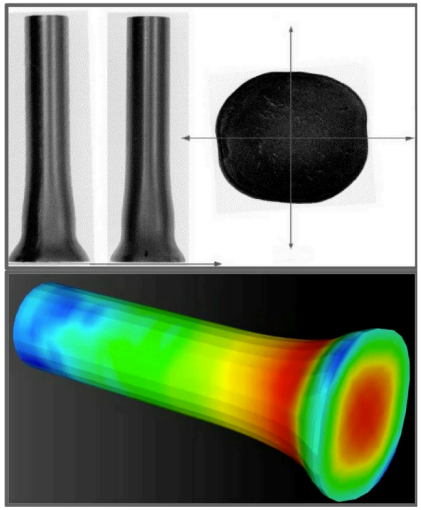
- A framework for multi-physics application development.
  - Control model
  - Execution model
  - Data model

Legion,
HPX, MPI

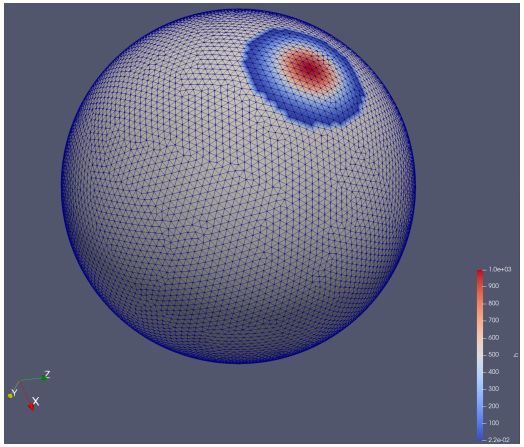
Los Alamos NATIONAL LABORATORY EST.1943




Runtime portability example, shock box



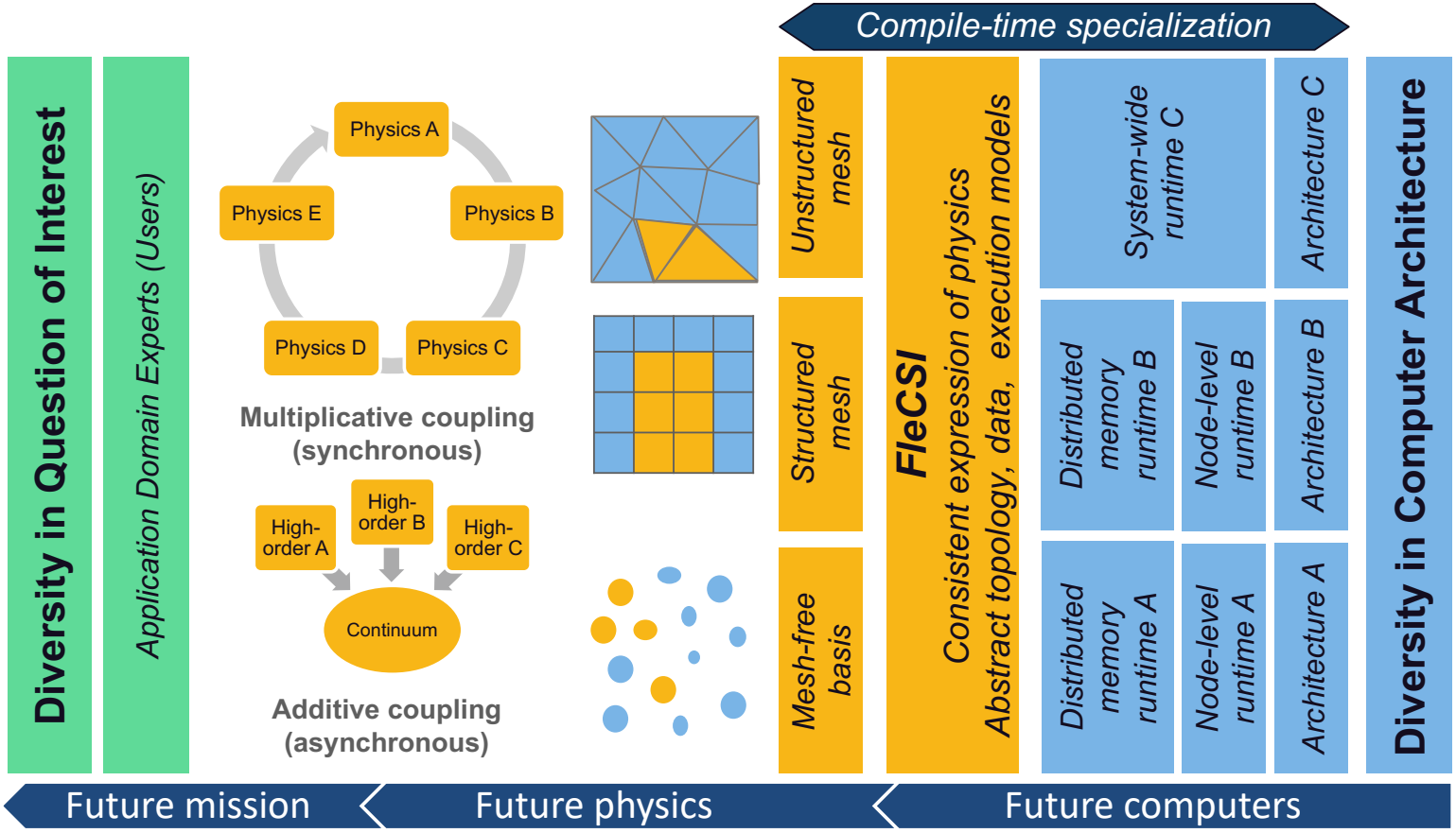
*Higher-fidelity predictive science through multiscale methods: A Flistra hydrodynamics code with an advanced grain-structure-aware material model (bottom image) captures the asymmetric deformation seen in Taylor anvil experiments measuring the behavior of metal under impact (top image). Courtesy: the Flistra project*



shallow water test case 1:
Advection of Cosine Bell over the Pole from the following reference.

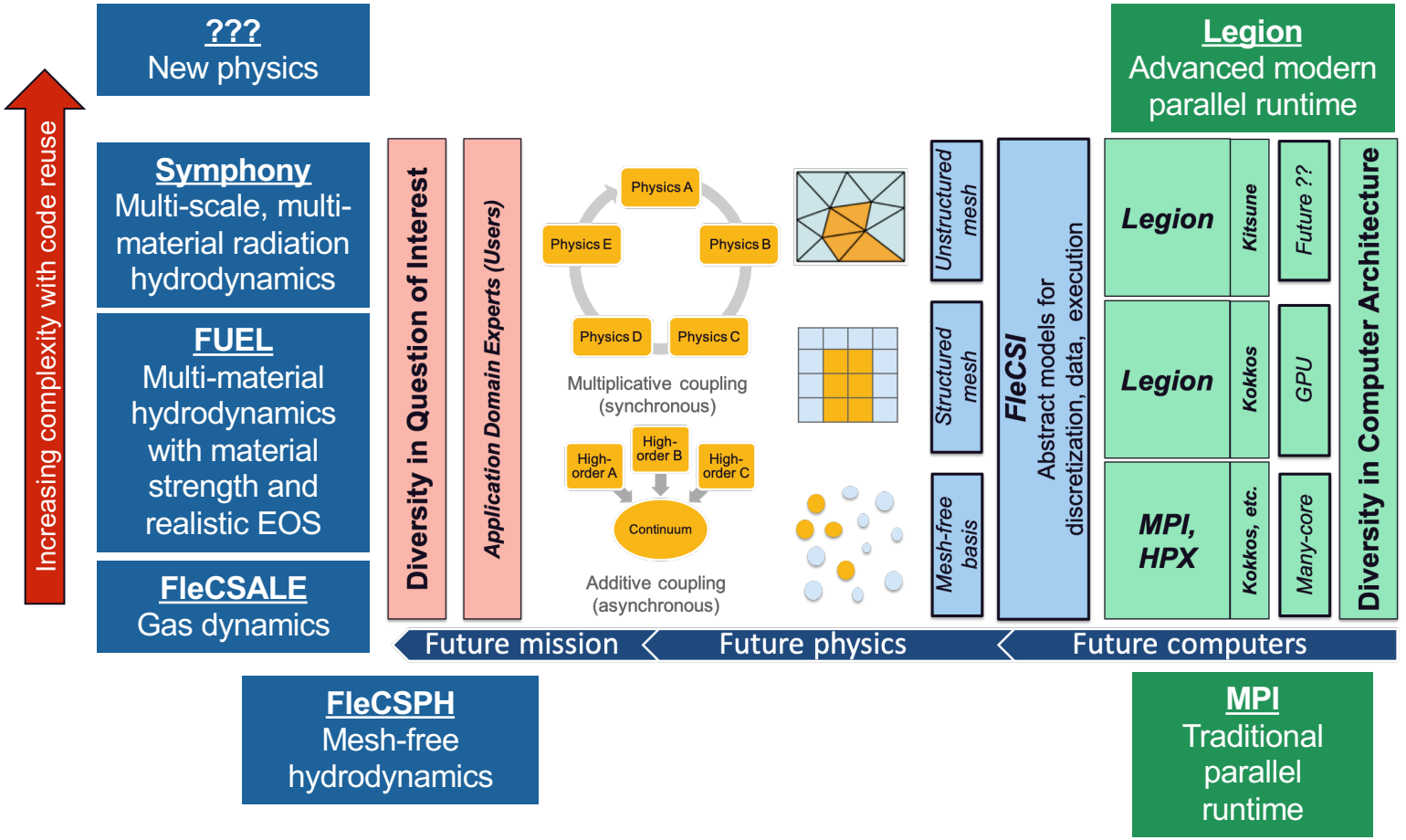
# Ristra goal: Multi-physics code developer productivity

*Realized through FleCSI, a compile-time configurable abstraction layer*



# Ristra SW Architecture: flexibility in a volatile future

*A variety of application codes have been developed using FleCSI*



# FleCSI primary goals

**Simplicity:**

Simplify multiphysics simulation development

Hide complicated logic on how to efficiently execute program on different HPC systems

**Abstraction:**

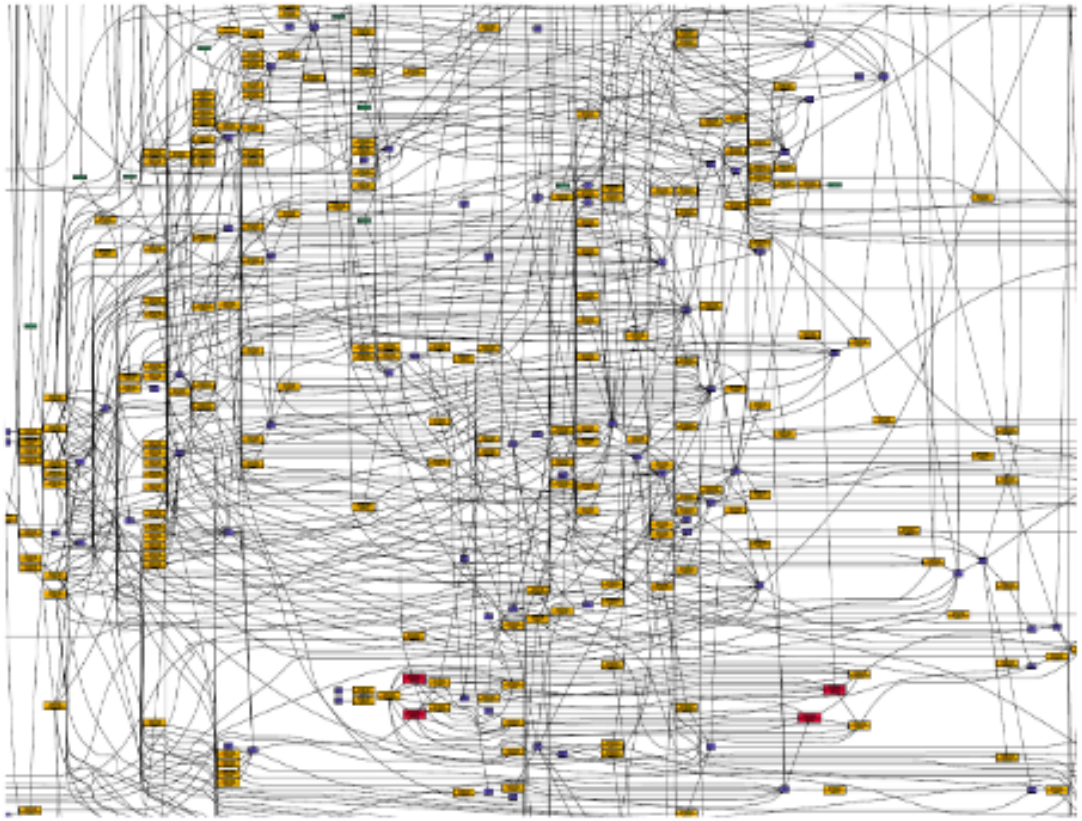
Abstract different run-times/programming models in a single interface

**Flexibility:**

Provide flexibility for target data structures/topology types

**Robustness**

# Why task-based runtimes?



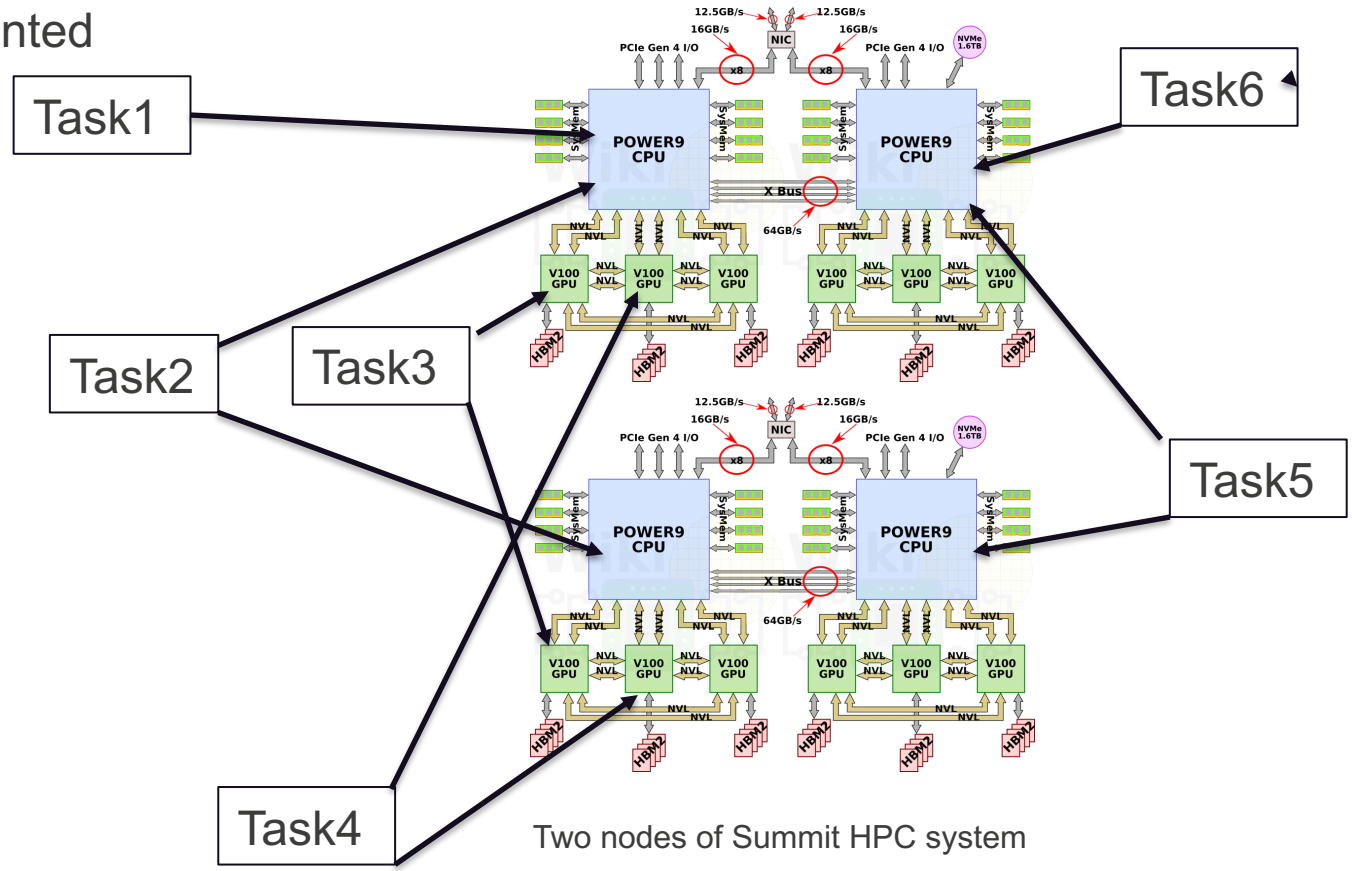
- Do you want to schedule that graph?
  (High Performance)
- Do you want to re-schedule that graph for every new machine?
  (Performance Portability)
- Do you want to be responsible for generating that graph?
  (Programmability)

MPI: programmer's responsibility

AMT: programming system's responsibility

# Legion

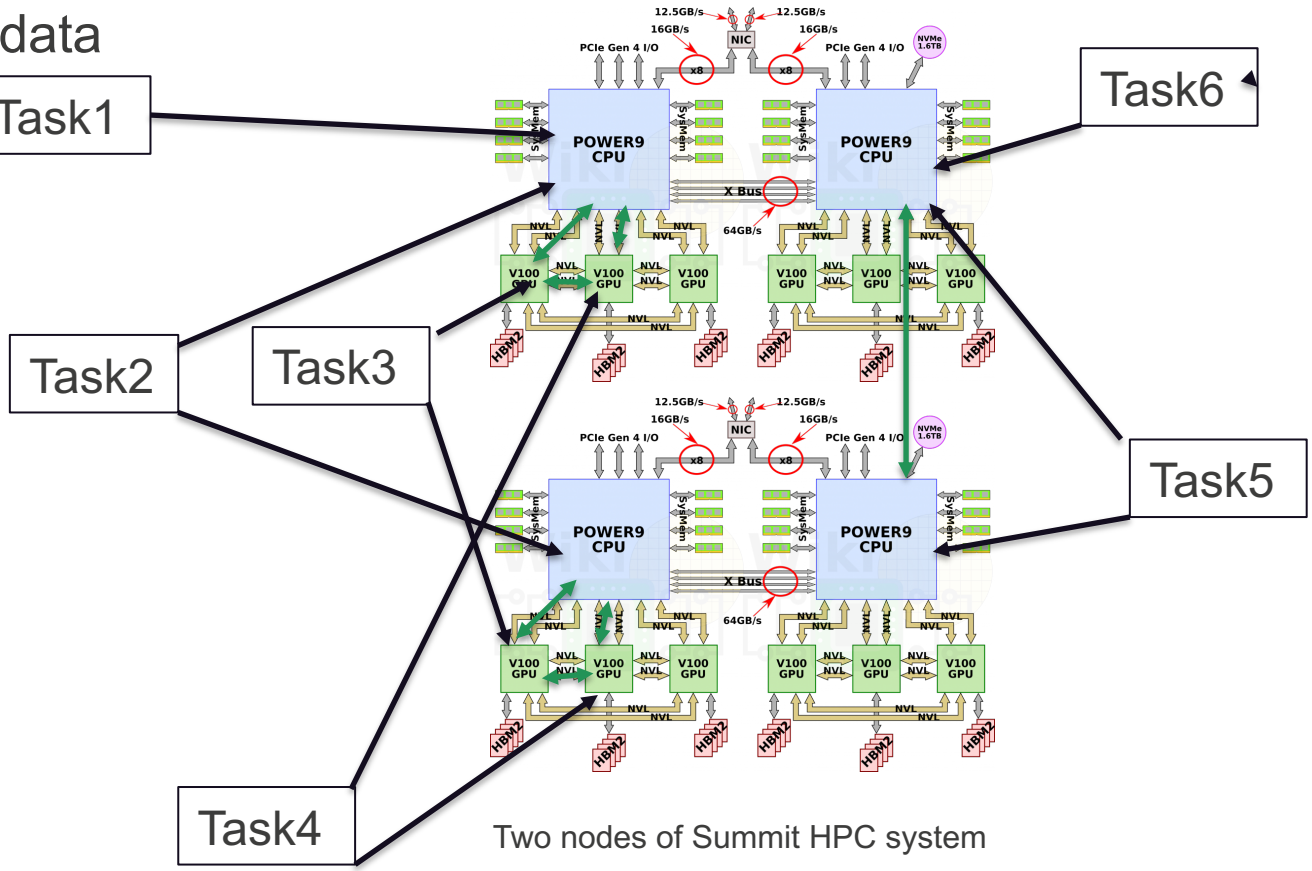
- Automated scheduling and latency hiding
  - Asynchronous tasking
  - Throughput-oriented




Two nodes of Summit HPC system

# Legion

- Easy access to GPUs
  - Simplifies programming complex hardware
- Easy control over data
  - partitioning
  - placement
  - layout in memory
- Kokkos support




Two nodes of Summit HPC system

# FleCSI API (tasks)

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  for (auto c : t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c){
      ...
    }
  }
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init>(burton, pressure);
  execute<check>(burton, pressure);
}
```

# FleCSI API

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  for (auto c : t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c){
      ...
    }
  }
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init>(burton, pressure);
  execute<check>(burton, pressure);
}
```

Creates partitioning of the mesh

Creates data containers (arrays, Logical Regions) and metadata for all entities of the mesh + connectivity information for burton_mesh_t type

# FleCSI API

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  for (auto c : t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c){
      ...
    }
  }
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init>(burton, pressure);
  execute<check>(burton, pressure);
}
```

Creates ‘pressure` field
on cells index space in a burton mesh

# FleCSI API

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  for (auto c : t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c){
      ...
    }
  }
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init>(burton, pressure);
  execute<check>(burton, pressure);
}
```

Execute FLeCSI task

# FleCSI API (tasks)

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  for (auto c : t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c){
      ...
    }
  }
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init>(burton, pressure);
  execute<check>(burton, pressure);
}
```

FleCSI task

# FleCSI API (tasks)

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  for (auto c : t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c){
      ...
    }
  }
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init>(burton, pressure);
  execute<check>(burton, pressure);
}
```

Data access permissions

- Used to create task dependency graph by Legion
- Used to reason about data exchange by FleCSI
- The dual use of the privileges is important

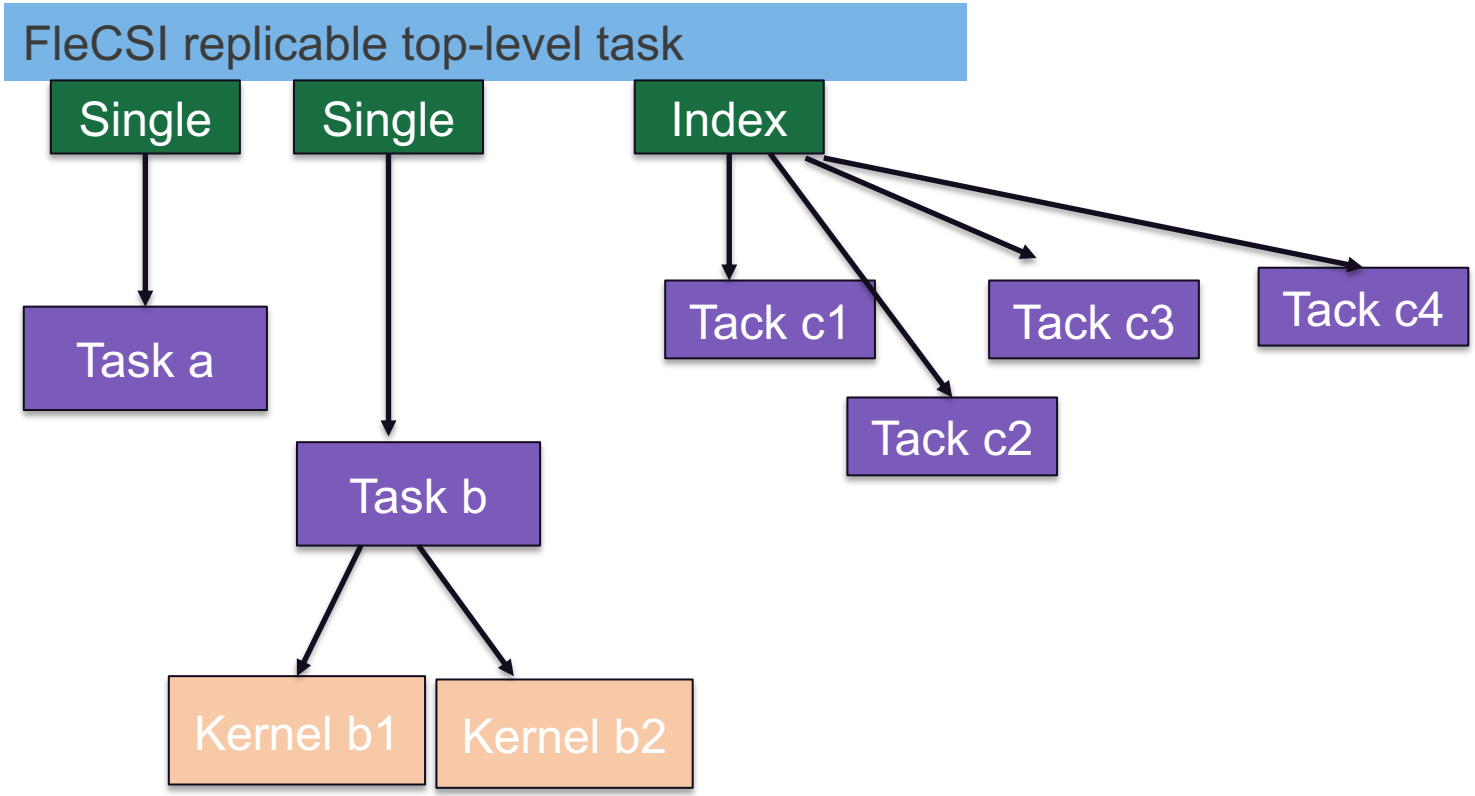
# FleCSI API (tasks)

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  for (auto c : t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c)){
      ...
    }
  }
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init>(burton, pressure);
  execute<check>(burton, pressure);
}
```

## FleCSI iterators

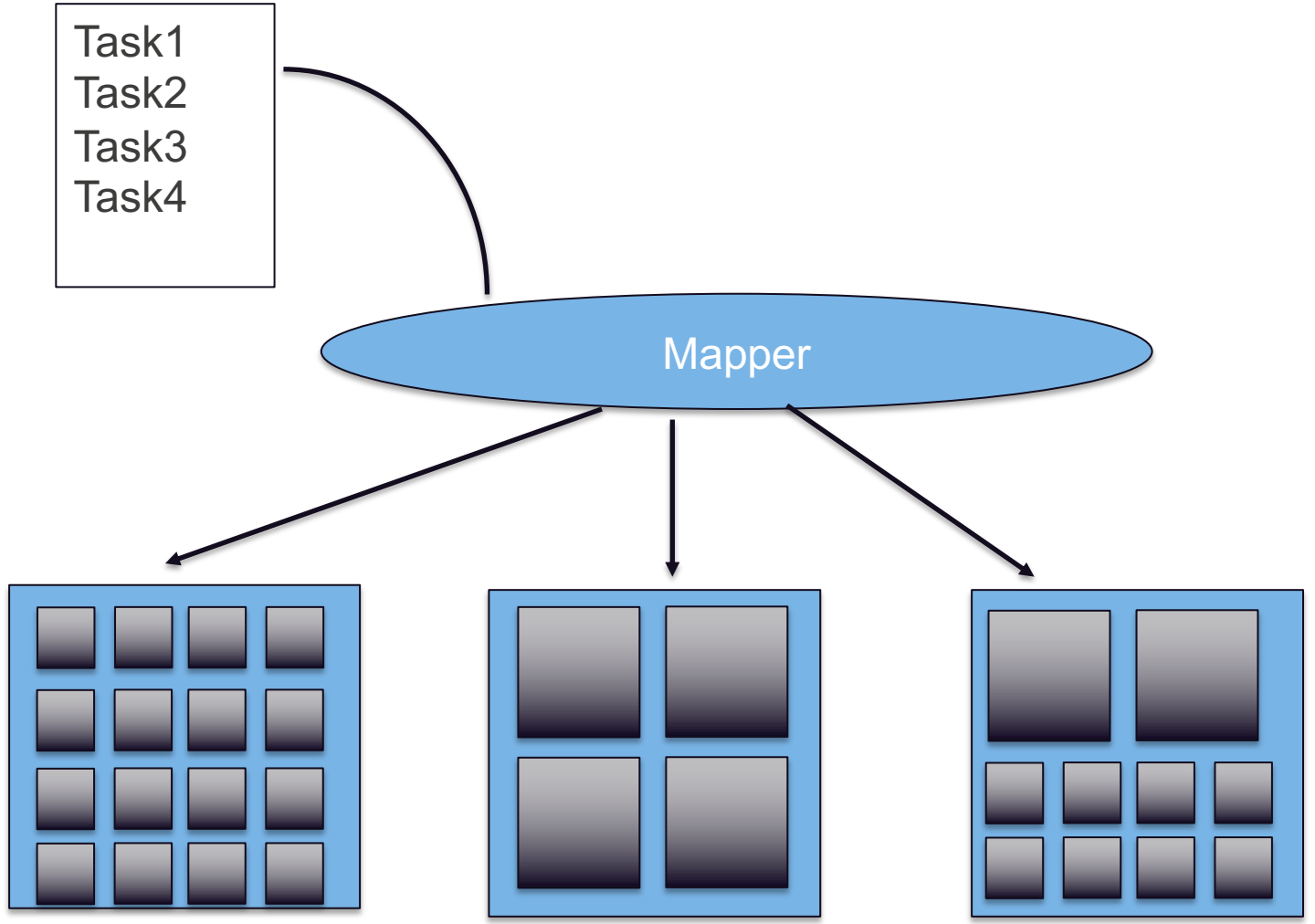
- Provide intuitive data access
- Work inside Kokkos

# FleCSI Execution model




Launch type

# FleCSI mapper



# FleCSI API ( tasks + kernels)

```
burton::slot burton;
burton::cslot coloring;

const field<double>::definition<burton, burton::cells> cell_field;
auto pressure = cell_field(burton);

void init(burton::accessor<wo> t, field<double>::accessor<wo> p) {
  forall (auto c, t.cells()){
    p(c) = 4.5;
    for (auto v: t.vertices(c){
      ...
    }
  };
} // init

void check(burton::accessor<ro> t, field<double>::accessor<ro> p) {
  for (auto c: t.cells()){
    assert(p(c) == 4.5);
  }
} // init

void driver() {
  coloring.allocate("input.txt");
  burton.allocate(coloring.get());

  execute<init, toc>(burton, pressure);
  execute<check, loc>(burton, pressure);
}
```

# FleCSALE

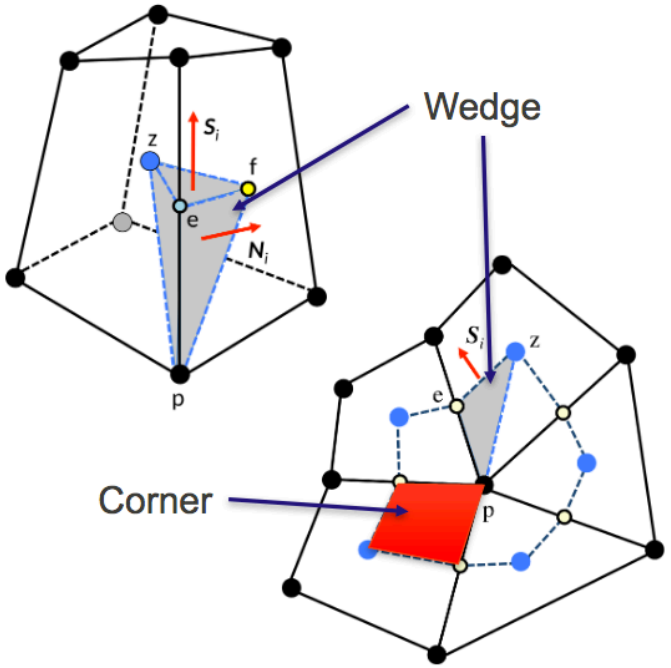
- FleCSALE - continuum dynamics software packages built on top of FLeCSI.
- Current status:
  - 2D/3D cell-centered Eulerian and Lagrangian solvers
  - 3D FEM Lagrangian solver

**Mesh is templated on dimension:**

**2D:**
```
burton_mesh_t<2> mesh;
```
**3D:**
```
burton_mesh_t<3> mesh;
```

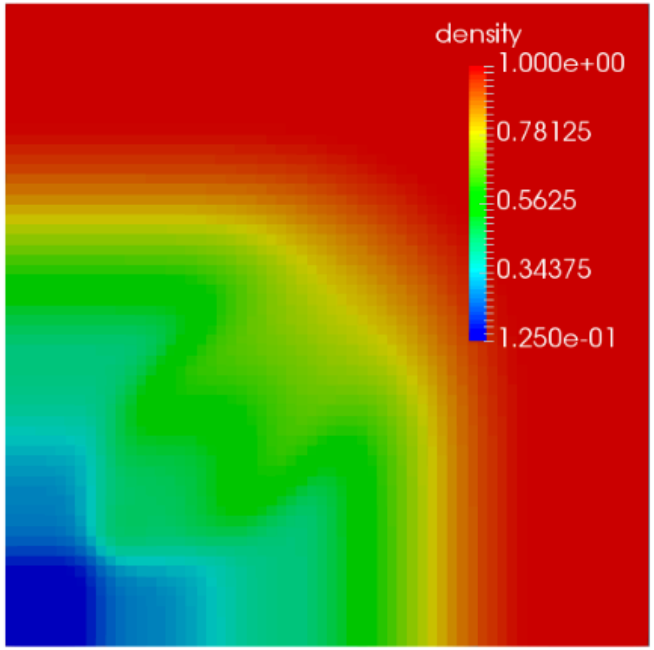
**Application code doesn't change (code works in 2D and 3D):**
```
for ( auto f : mesh.faces() )
  auto n = f->normal();
  // do some work
```

**Mesh has wedges and corner data structures in addition to vertex, edge, face, and cell primitives:**
```
for ( auto cn : mesh.corners() )
  for ( auto wg : mesh.wedges(cn) )
    auto n = wg->facet_normal();
    // do some other work
```

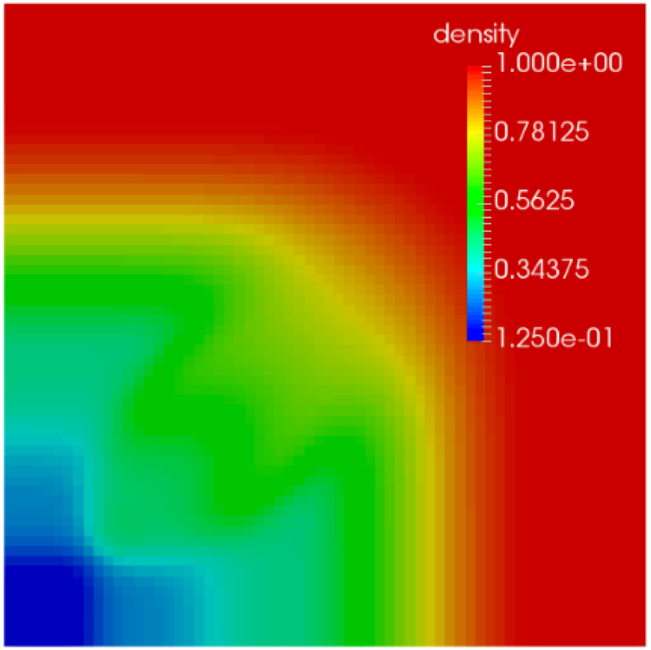


## FleCSALE Runtime Portability

**FleCSI allows switch between MPI and Legion runtime models with no change to application code**

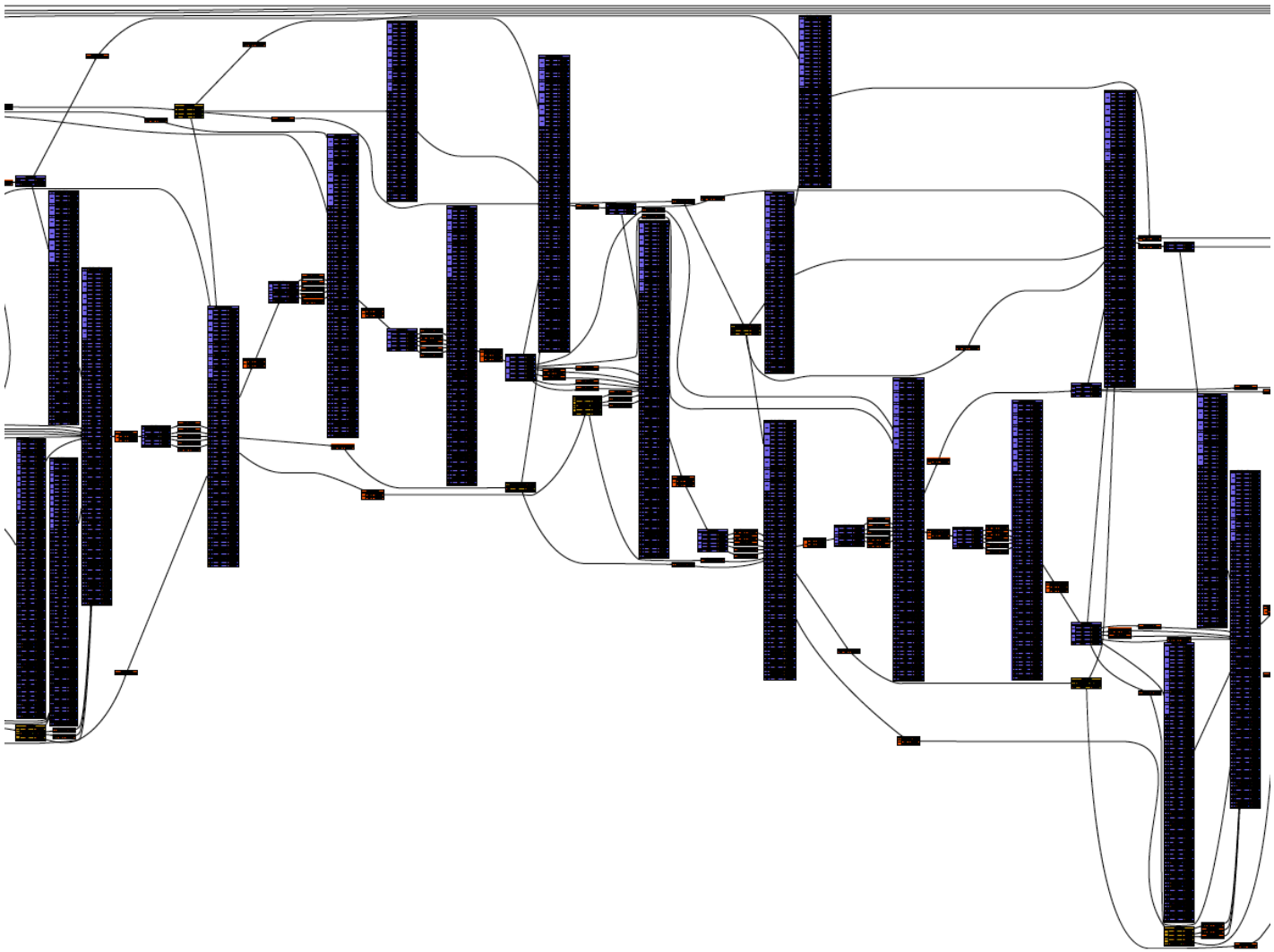



MPI

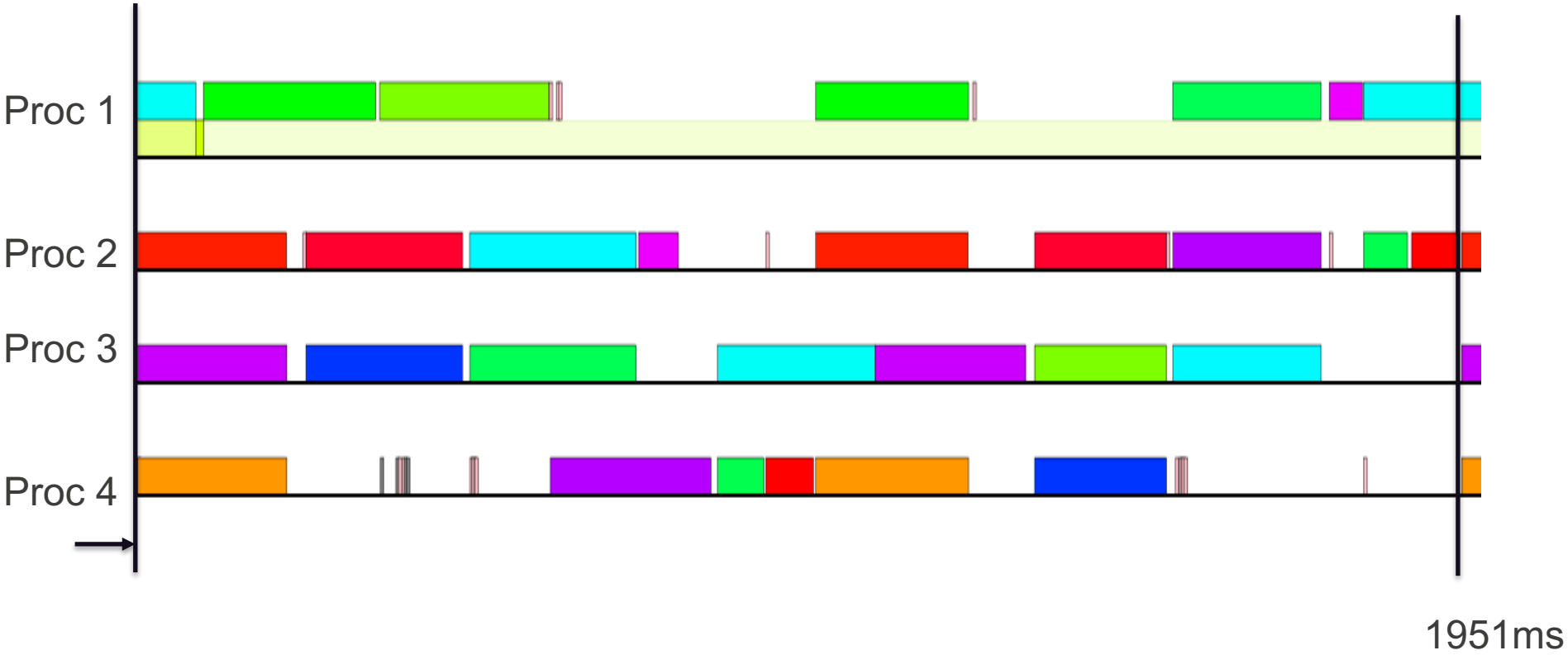



Legion

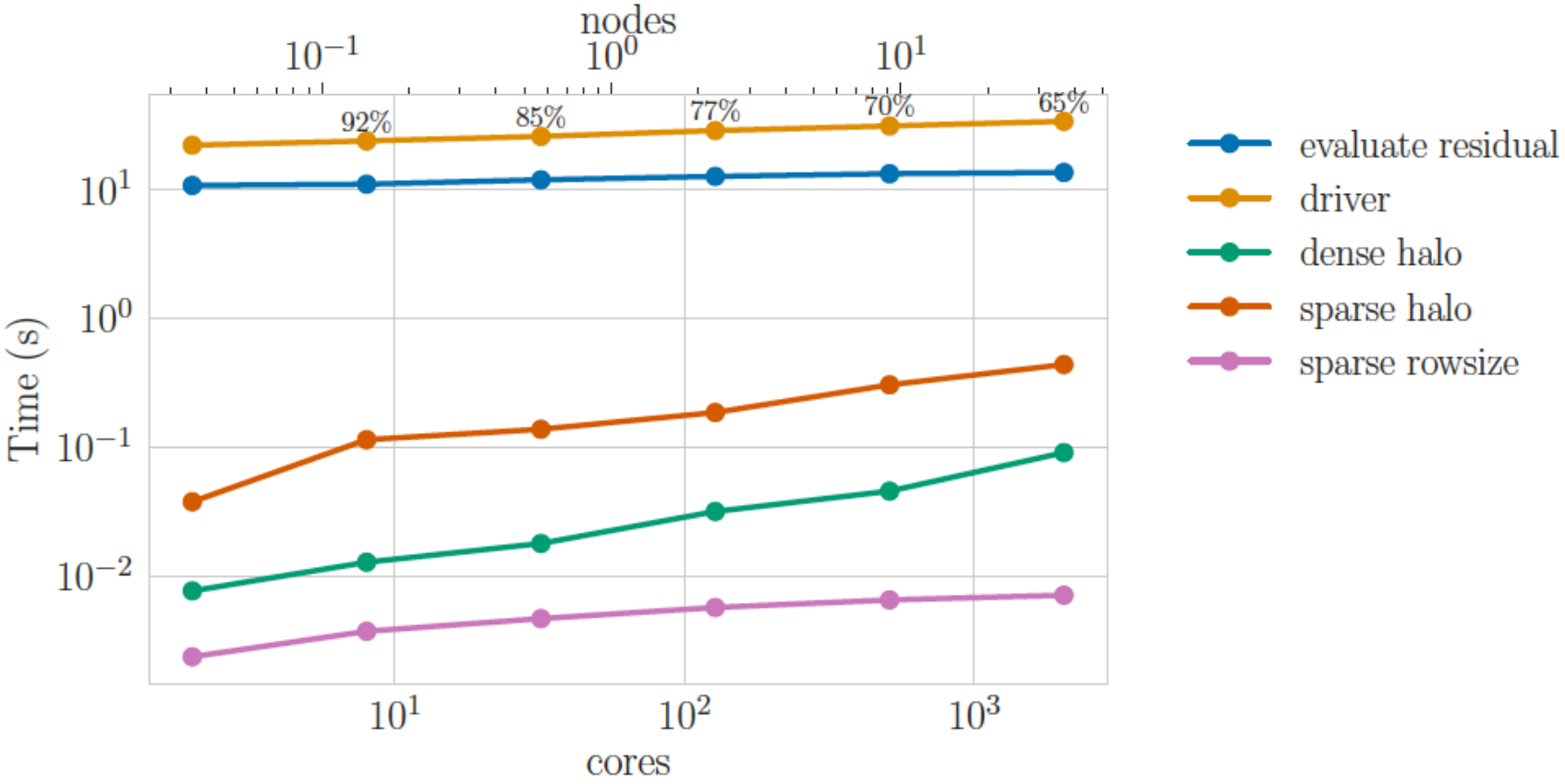
# Profiling results (FleCSALE, hydro_2d)



# Legion Prof output



# Weak Scaling results (maire_hydro_2d, MPI)

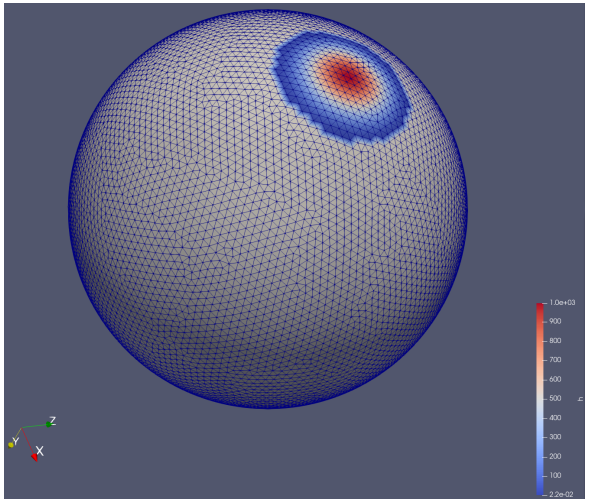


# MPAS-Ocean

MPAS-Ocean is designed for the simulation of the ocean system from time scales of months to millenia and spatial scales from sub 1 km to global circulations.

***Current status*** of MPAS-O-FleCSI:

- advection – diffusion application
- shallow water application



shallow water test case 1: Advection of Cosine Bell over the Pole from the following reference.

**CANGA**
**Coupling Approaches for Next-Generation Architectures**

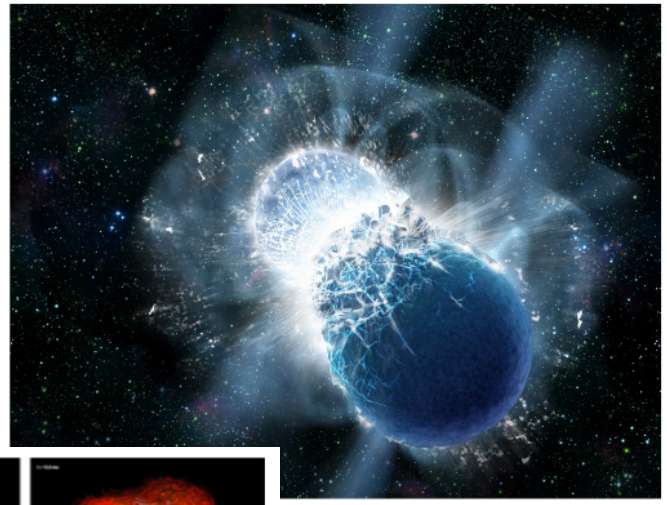
# FLeCSPH

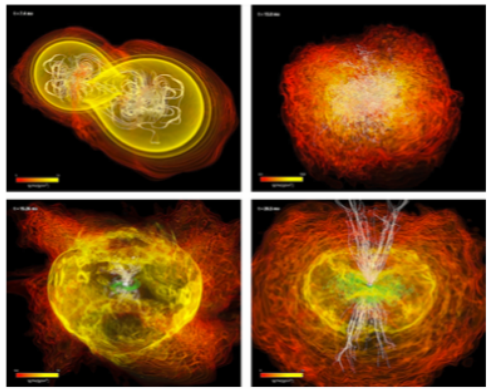

smoothed-particle hydrodynamics (SPH) solver for the solution of Lagrangian problems in astrophysics and cosmology.

**Uses FleCSI tree topology**

**Initial focus on astrophysics simulations of neutron star merges**

Potential source of gravitational waves, macronovae, and nucleosynthesis





Rezzola et al. (2011)


: Performance Portability Programmability

Would like to find more ?

https://flecsi.org/